IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of		)	Art Unit:	2191
	Iborra and Pastor	)	Examiner:	Rampuria
Serial No.:	09/872,333	)	Docket No:	CHG-001.3P
Filed:	06/01/01	)		
For:	AUTOMATIC SOFTWARE PRODUCTION SYSTEM			

Commissioner of Patents
P.O. Box 1450
Alexandria, Virginia

INFORMATION DISCLOSURE STATEMENT

Sir:

Pursuant to 37 C.F.R. §§1.97-1.98, the undersigned would like to make the following prior art references of record in the above-identified patent application. The undersigned believes that some of these references may be material to the examination of this application and in respect of which there may be a duty to disclose in accordance with 37 C.F.R. §1.56. English language summaries of the Spanish language papers which have been received from the client are included herewith.

Many of these references were disclosed in the parent patent application entitled AUTOMATED SOFTWARE PRODUCTION SYSTEM, serial number 09/543,085, filed 4/4/2000 (the parent case CHG-001, now US patent 6,681,383) so paper copies of those references are not included pursuant to 37 CFR Rule 1.98(d), however PDF copies of these papers are supplied on the CD-ROM included with the IDS previously submitted dated 5/26/05. Copies of the 1449 forms from the parent case office actions are included with the office action copies of the office actions from the parent case on the enclosed CD-ROM. The previously submitted CD-ROM Volumes 1 and 2 (all four volumes are on the CD-ROM previously submitted) contains papers most of which are prior art and which were published by some of the inventors at various times. If a publication date is not included in the description of the paper below, no date information is available and applicant's do not admit the paper is prior art until further evidence surfaces. Volumes 3 and 4 of the enclosed CD-ROM contain office actions from related US cases (Vol. 3) and related foreign cases (Vol. 4).

There is new Federal Circuit case law holding it to be inequitable conduct to not disclose office action rejections of similar claims in related cases. Accordingly, copies of office actions from related cases are included herewith on the CD-ROM previously submitted. This family of cases comprises the parent case (CHG-001 folder on the CD-ROM Vol 3) recited above which claims an entire system and three CIP cases (CHG-001.1P, CHG-001.2P and CHG-001.3P folders on the CD-ROM Vol. 3) which claim various subsystem combinations of elements of the system of the parent case. There is also a PCT case filed from the parent case (IPER and search report, and Canadian, Japanese, European and Australian cases stemming therefrom (office actions, IPER and Intnl Search Report included herewith on Vol. 4 of the CD-ROM previously submitted). There is also another case entitled METHOD AND APPARATUS FOR AUTOMATIC GENERATION OF INFORMATION SYSTEM USER INTERFACES, serial number 10/356,250, directed to the graphical user interface which has not had any office actions yet. Copies of the 1449 Forms submitted in these CIP cases are submitted herewith as attachments to the office actions from the related cases on the previously submitted CD-ROM Vol. 3.

While this Information Disclosure Statement may contain material information pursuant to 37 C.F.R. §1.56, it is not intended to constitute an admission that any individual reference referred to herein is prior art to the invention disclosed and claimed in the above-identified patent application.

Each reference listed herein may be accompanied by an explanation of its relevance or of the teachings of the reference. The IDS submitted on 5/26/05 did not have explanations of all the references as that work was ongoing. This IDS is more complete in terms of explanations of references. While these explanatios are believed to generally reflect the contents of the references, copies of the references are included on the previously submitted CD-ROM so the Examiner can draw his own conclusions. The undersigned does not know whether a reasonable examiner might consider any of the attached references relevant and material to the examination of the above-identified patent application. Therefore, it is not intended that the examiner rely on the description as unfailingly accurate or complete, and the Examiner should make his own inspection of each reference.

A copy of each reference cited herein in PDF or TIFF Group 4 compressed format on CD-ROM was enclosed with the 5/26/05 IDS for the express purpose of providing the examiner with an opportunity to perform an independent evaluation to arrive at an independent assessment of its relevance and materiality, if any, to the claimed subject matter. Where available, complete or partial English language translations of the pertinent portions of non English language papers are supplied herewith. To the extent they are not supplied here, they will be supplied in a supplemental IDS when received from the client. Summaries of some of the papers have not yet been prepared as not all papers have been read yet.

Cited Art

1) Letelier, P., et al., "*OASIS Version 3.0: Un Enfoque Formal Para el Modelado Conceptual Orientado a Objectos*" (In Spanish), ISBN: 84-7721-663-0, Legal Diposit: V-3484-1998, Servicio de Publicaciones de la UPV, SP-UPV 98-4011, Valencia, Spain, 1998.

2) Pelechano, V., "*OO-Method: Implementación de un Entorno Gráfico para el Análisis y Diseño de Sistemas De Información OO*" (In Spanish), Master Thesis, 1994.

3) Pelechano, V., et al., "*CASE OO-Method: Un Entorno de Producción Automática de Software*" (In Spanish) Actas de la Convenció Informática Latina CIL-95, Barcelona, June, 1995.

4) Romero, J., "*Diseño de un Entorno de Producción de Software basado en el Lenguaje de Especificación OASIS y en la utilización de PowerBuilder como Herramienta de Desarrollo Gráfica y C/S*" (In Spanish), Master Thesis, Valencia, March, 1996.
Reference:
[Romero96MT]

Original title:
"Diseño de un Entorno de Producción de Software basado en el Lenguaje de Especificación OASIS y en la utilización de PowerBuilder como Herramienta de Desarrollo Gráfica y C/S"
Title translation:
"Design of a Software Production Environment based on the OASIS Specification Language and the use of PowerBuilder as Client/Server and Graphical Development Tool"
Summary of contents:
This master thesis is about transforming an OASIS formal specification to a computer program written in the PowerBuilder language and running on a Microsoft Windows operating system. The core of this work thus establishes a relationship between OASIS, Windows and PowerBuilder.

Following is a description of the contents of the different sections the work comprises:

Section One
Introduction to the work, its goals and how its contents are structured.

Section Two
Sets the grounds for the work.
First, it describes the elements in the OASIS v2.2 formal specification language used in the work:
- Valuations
- Derivations
- Integrity constraints
- Preconditions
- Triggers
- Processes
- Complex classes

The formal language specification written in OASIS is provided in a textual notation, unlike the graphical means provided by the patented invention which allows the user to input the functional requirements for the application to be developed in a graphical way without the need to know the syntax of the underlying formal specification language (OASIS) into which the input data is automatically transformed.
Every conceptual primitive (valuations, derivations, ...etc) is described here using dynamic logic, which defines the underlying semantics of each primitive, but it does not provide precise detail about what can be expressed with each conceptual primitive
(e.g: it says that a valuation is a formula with the form _ [a] _' and explains what this means in the context of dynamic logic, but it does NOT say that a valuation is a condition-effect pair, nor that there are three types of valuations).
Apart from this another remarkable difference with the patented invention is that it does NOT teach the use of a presentation model, because at that time (1996) the Presentation Model did not exist yet as part of our formal specifications.

Then it introduces how the Client/Server architecture is implemented on the Microsoft Windows operating system.

Last it presents the relevant features of the PowerBuilder product.
Section Three

Describes how the elements in the OASIS formal specification are mapped to elements in both

Windows and PowerBuilder. It does NOT teach, however, any method to automate the transformation of an OASIS formal specification into Windows GUI elements not PowerBuilder elements.

First it describes how elements in the OASIS formal specification are mapped to elements in the Windows GUI. Specifically, it teaches how the application menu is created from the set of classes, and services of said classes and relationships between the classes (but it does NOT teach how to create a menu from anything coming from a Presentation Model).

Then it describes how elements in the OASIS formal specification are mapped to elements in PowerBuilder. Here we distinguish three types of mappings:

- To create the data model
  - o Teaches how classes, attributes and relationships are mapped to tables, columns, primary keys and foreign keys in a relational database.
- To create the process model
  - o Teaches how events and valuations are mapped to PowerBuilder functions. Creation and destruction events are mapped to "INSERT" and "DELETE" SQL statements executed directly on the database while other events with valuations are mapped "SELECT" SQL statements to read the actual values of attributes, functions to modify the values of attributes with the value of arguments of the event then to "UPDATE" SQL statements that use the new values of attributes to update a tuple in a table in the database.
  - o Teaches how preconditions, integrity constraints and triggers are mapped to functions to check them.
- To create the protection model
  - o Teaches how to create code to control that every user accesses only the information he is allowed to access according to his privileges.

5) Pastor, O., et al., "*An Object Oriented Methodological Approach for Making Automated Prototyping Feasible*", Database and Expert Systems Applications, Lecture Notes in Computer Science (1134) pgs. 29-39 Springer-Verlag, 1996, ISBN: 3-540-61656-x, ISSN: 0302-9743, Zurich (Suisse) September 9, 1996 [Pastor 96aO]: Teaches a method of building a formal language specification in a formal language called Oasis of a program to be automatically written. This is done by defining graphic models in an analysis phase where the program requirements are analyzed, to wit: an object model, a dynamic model and a functional model are built. The object model defines classes of objects. Each class has a signature including attributes, methods and actions and a set of formula of different kinds to define integrity constraints, conditions that must be satisfied, valuations whcih explain how attributes are changed by events, derivations which relate some attribute values to others, preconditions which determine when an event can be activated, and triggers which introduce internal system activity. The formal language specification is generated in an automated way from the three models to generate a complete system repository where all the relevant properties of the component classes are included. A prototype is then built in an automated way from this formal language specification with two translations: one from the models to an Oasis language specification; two, from Oasis to the selected programming environment. The object model defines aggregation and inheritance hierarchies and agents are introduced to specify who can activate each class service. The dynamic model specifies valid object lives using state transition diagrams and interobject interaction using an object interaction diagram. The functional model captures semantics attached to any change of state of an object. The main idea is to specify how event

activation in a given state changes the attribute values. The model specifies how an event changes the values of the relevant attributes through an interactive dialogue. The value of every attribute is modified depending on the action that has been activated, the involved event arguments and present object state. A translator converts the graphic information of the models into Oasis language specification statements. Whole system classes are converted into elementary Oasis classes where attributes are declared (constants, varialbe and derived), static integrity constraints are declared, derivation formulas for derived attributes and private and shared events are declared. Trigger relationships, event preconditions, process definitions attached to any class and global interactions are obtained from the dynamic model. Preconditions appear as transition labels on the state transition diagram (STD), and process definition is achieved from the paths in the STD. Trigger and global interaction sections of the Oasis specification are obtained in an automated way from the object interaction diagram. Valuation formulas are derived through dialog boxes the analyst uses during the analysis phase while generating the functional model.

6) Pelechano, V., et al., *"Implementación y comprobación de restricciones de integridad dinámicas en entornos de programación orientados a objectos"* (In Spanish), II Jornadas Nacionales de Ingeniería de Software, Universidad el País Vasco, San Sebastián, 3-5 Septiembre 1997, pgs. 101-117.

7) Pastor, O., et al., "*Linking Object-Oriented Conceptual Modeling with Object-Oriented Implementation in Java*", VIII Conference on Database and Expert Systems Applications, (DEXA'1997), ISGN: 3-540-63478-9, LNCS (1308), Toulouse, France, 1997 [Pastor 97a]:
Teaches the OO-Methodology to allow analysts to introduce relevant system information defining a computer program to be automatically written by means of a set of graphical models to obtain a Conceptual Model of the program. This allows a formal language specification to be written at any moment. A Java prototype which is functionally equivalent to the OASIS formal language specification can be automatically generated from the formal language specification by defining an execution model that gives the pattern to obtain a concrete implementation in the selected target software development environment. This is done by precise mapping between formal specificaton concepts and Java components that implement them. Section 4 of the paper describes how the OO-Method models developed by the analyst are converted to a Java program. Teaches an object model, dynamic model and functional model. Implementation uses Java as a programming language and a relational database as a persistent object repository. The process of generating JAVA code involves binding the execution model to specific features of te selected programming environment. The Execution model has three main steps: access control; object system view; and service activation. Any service execution invovles: object identification; introduction of event arguments; state transition correctness; precondition satisfaction; valuation fulfillment; integrity constraint checking in the new state; and trigger relationships testing.

8) Pastor, O., et al., "*OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods*", 9th International Conference on Advanced Information Systems Engineering, (CaiSE'1997) ISGN: 3-540-63107-0, LNCS (1250), Barcelona, Spain, June 16, 1997 [pastor97b]:
Teaches an approach to computer program prototyping using a combination of conventional OO methodologies coupled with an OO formal specification language and use of a graphical OO conceptual modelling environment which collects the system properties considered relevant for building a a formal, textual OO specification in an automated way. The conventional OO

methodology covers: classes and objects; abstraction; encapsulation, inheritance and aggregation to deal with complex classes; and interobject communication. A CASE workbench with supports this working environment was said to exist, and it was not limited to generating only static templates for the component system classes. Teaches use of a conceptual model (object model, dynamic model and functional model) to capture the program requirements. A formal language Oasis specification is automatically generated from the conceptual model, and acts as a complete system repository of all relevant properties of the component classes. An execution model is used for automated prototyping. The details of the object model, dynamic model and functional model are explained.

A code generation strategy implements a mapping between the conceptual model and the execution model. The execution model accurately states the implementation-dependant features associated with the object society machine representation. The execution model explains the pattern to be used to implement object properties in any target software development environment. The execution model implements access control, object system view and service activation. A sequence of actions is defined to identify a server object, introduce event arguments, etc. Following these steps during implementation of the program assures the functional equivalence among the object system description collected in the conceptual model and its codification in the target programming environment.

A code generation strategy is introduced, and recognition is made that once an execution model is introduced, there are different concrete implementations of the same execution model in different software developement environments. The code generation strategy is to reproduce the user's mental image of the system within the OO world view such that the system is represented as a a society of interacting objects where every individual object can access other system component objects and activate those services it is allowed to invoke. Consistency in presentation (what the user sees), behavior (how the application reacts) and sequencing (how the dialogs are sequenced) and functionality (how actions are carried out) is the goal. Finally, the system code must give control to the user and not control the user.

Both a static and dynamic point of view are dealt with and the static point of view defines the relational database schema.

An access control window, main window, event window and observation window are taught as interfaces to operation of the automatically generated program. The log in window controls user access. The main window presents the system view, the event window allows corresponding arguments to be introduced and the observation window allows the user to see the results of any query done over the current object state.

The OO Method CASE tool is described in section 4. The purpose of this tool is to simplify the analysis, design and implementation of the desired program from an object oriented perspective. The CASE tool provides the ability to generate code in a well-known industrial software development environment from the system specification. Analysts collect information and can geneate a formal OO system specification, and a complete (including static and dynamic) software prototype which is functionally equivalent to the quoted system specification can be generated anytime the analyst so desires. The CASE tool, when started presents a blank blackboard that represents the CCD where the analyst can draw classes and their properties. By selecting one of the classes on the CCD, the user can change to the dynamic model. the OID completes the dynamic model. A menu option called Design generates Oasis code in an automated way and other menu options allow automated code generation in C++, Delphi or Java.

9) Pastor, O., et al., "*Object Oriented Conceptual Modeling Techniques to Design and Implement a Sound and Robust Oracle Environment*" Actas de Oracle OpenWorld 97, Viena (Austria) 7-11 Abstract publicado en Oracle OpenWorld Review pg. 36, April, 1997 [Pastor 97c0).
Teaches a precise mapping between the object oriented notions of the conceptual model to ORACLE structures for implementation at the programming level. The conceptual model includes an object model, dynamic model and functional model. The execution model is a generic pattern applicable to any conceptual model. The translation into Oracle structures has two main phases: the relational logic model; and the packages, procedures, functions, triggers and forms needto to introduce the dynamic into the system. To obtain the relational logic model, the attribute definitions and static integrity restrictions are focussed upon.

10) Romero, J., et al., "*Una Herramienta de Generación Automática de Software*" (In Spanish) In Procs of IDEAS-98 - I Workshop Iberoamericano en Ingeniería de Requisitos y Ambientes Software, Porto Alegre, Brasil, April 1, 1998 [romero98]:

Original title:
"Una Herramienta de Generación Automática de Software"
Title translation:
"A Tool for Automatic Code Generation"
Summary of contents:
This paper describes the main features of a CASE tool to support the OO-Method (the methodology whose formal background is the OASIS formal specification language), the elements in an OASIS formal language specification and their correspondences in the OO-Method Execution Model.

The paper states that the CASE tool automatically produces code so as to allow rapid prototyping of applications (it does not teach the automatic production of code for full applications, just for prototypes).

Describes the different elements in an OASIS formal language specification from the dynamic logic point of view.
With respect to valuations they are not categorized as in the patented invention. Instead, it is attributes that are categorized. This is an idea that was later abandoned because it proved useless and impractical.

Teaches that the OO-Method divides a formal language specification (Conceptual Model) into three complementary views: Objects Model, Dynamic Model and Functional Model. No mention to the Presentation Model is made (it had not been defined yet).

Then it teaches that the Execution Model comprises seven steps:

- Identification of the object providing a given service.
- Introduction of service arguments.
- Check correctness of current state of the object.
- Check that preconditions associated to the service hold.
- Perform calculation associated to valuations
- Check integrity constraints

- Check trigger relationships

Finally presents two portions of C++ code allegedly produced by some automatic translator.

In the conclusions section, the paper acknowledges that no solution had been found yet to generate code when aggregation and inheritance relationships were present in the formal language specification.

11) Gomez, J., et al., "*The Execution Model: A Component-Based Arquitecture to Generate Software Components from Conceptual Models*" In Procs of International Workshop on Component-based Information Systems Engineering, 10th International Conference on Advanced Information Systems Engineering, CAiSE-98 Pisa (Italia), pgs. 87-94, ISSN 1170-487X June 8, 1999 [gomez98]:
Teaches how industry attempts to provide unified notations such as UML for Object Oriented methodologies have led to an excessive set of models and overlapping semanticss without a methodological approach. Rational Rose and Paradigm Plus are mentioned as CASE tools which include code generation from the analysis models they can be used to generate. However, further inspection indicates it is not at all clear how a final software product which is the functional equivalent of the system description collected in the conceptual model can be generated. Rational Rose results in a template for the declaration of classes where no method is implemented and no related architectural issues are taken into account. The paper presents a component based architecture based on a formal object oriented model which gives a pattern for obtaining software components. These components set up a basis for a software prototype which is functionally equivalent to the conceptual model. The models used to capture the Conceptual Model are described: the object model, the dynamic model; the functional model.

The execution model is then introduced. The execution model accurately states the implementation dependent features associated to the selected object society machine representation. An object society is introduced where an active object is immersed in the object society as a member and interacts with th eother society object. To achieve this behaviour, the system has to: identify the user by requiring the user to log on and provide the object system view which determines the set of object attributes and services that the user can see; allow service activation of services of objects the user can view (any service activation has two steps: build the messages and execute it); identify the object server; introduce event arguments; check state transitions; determine precondition satisfaction; valuation fulfillment; integrity constraint checking in the new state; and trigger relationship tests. A component based architecture for the execution model includes a persistence tier, an interface tier, and an application tier.

12) Pastor, O., et al., "*From Object Oriented Conceptual Modeling to Automated Programming in Java*", Conceptual Modeling — ER'98, Lecture Notes in Comptuer Science (1507), pgs. 183-197, Springer-Verlag, 1998, ISBN: 3-540-65189-6, ISSN: 0302-9743, Singapur, November 16, 1998 [pastor98c0]: Teaches the OO-Object model to graphically define object model, dynamic model, and functional model that together comprise a Conceptual Model of the program to be automatically written and translating these models automatically to an OASIS formal language specification. Also teaches an execution model which identifies the system user and determines the set of object attributes and services that the user can see and activate. After connection and establishment of the user's world view, the user can activate services. Service activation involves two steps: building the service activation message; and execution of the message. To build the message: 1) the object server associated with the desired service to be executed is identified; 2) the event arguments are asked for by the interface tier. Once the message is sent, the service execution by the server object is characterized by the occurrence of the following sequence of events: 1) check state transitions; 2) check preconditions, and, if either 1 or 2 is not satisfied, ignore the message and do not activate the service; 3) valuation formula fulfillment; 4) integrity constraint checking; 5) trigger relationship checking. This paper also mentions a tool called IPOST that automatically generates a prototype from an object-oriented analysis model. This paper also mentions translating a Conceptual Model into Java classes using the Execution

model by defining the architecture of the classes needed to implement the three tier architecture of the Conceptual Model: the interface tier, the application tier and the persistence tier. Teaches at the interface level definition of a Java access control class which extends a panel with the typical widgets to allow a user to be identified and determine the user's password and class of users to whom he or she belongs. Teaches a system view class which has an instance or object for every active user and displays what objects and services the user is allowed to see and access. Teaches a service activation class which defines a typical web interace for data entry. At the application level, teaches classes that implement the behavior of the business classes. The Java business classes are defined as an implementation of OASIS itnerface (to specify the necessary services to support the execution model structure) and an extension of an object_mediator class. At the persistence level, an object_mediator class is created. It implements the methods for saving, deleting and retrieveing system domain objects that are stored ina persistent secondary memory object repository.

13) Pastor, O., et al., "*Mapping Aggregation from Object-Oriented Conceptual Modeling to Object-Oriented Programming*", In Procs of Third International Conference on Object-Oriented Technology, WOON-98, pgs. 59-70, San Petersburgo, Russia, July 2, 1998 [pastor98do]
Teaches implementing the aggregation concept in an object-oriented environment, and, in particular, an aggregation concept provided by the formal language OASIS. The paper studies the aggregation concept in OASIS and another specification language called Troll and describes how to implement the aggregation concept in the OO-method based upon OASIS. Implementation using data models and patterns of cardinalities are taught.

14) Romero, J., et al., "*Automatic Object-Oriented Visual Programming with OO-METHOD*", Software and Hardware Engineering for the 21th Century, pgs. 345-354, World Scientific and Engineering Society Press. ISBN: 960-8052-06-8, July 1999 [romero99]:
This paper deals with the problem of creating efficient automatic code generators from an object-oriented conceptual model. The OASIS formal object-oriented specification language is described. Various concepts of OASIS classes are introduced including: inheritance; aggregation; shared events; triggers; global interactions. The conceptual model is introduced as being comprised of an object model, a dynamic model comprised of a state transition diagram and an object interaction diagram; and, a functional model. An execution model is then introduced. The execution model sets the implementation details for the automatically generated code in order to determine user interface, access control, service activation, etc. Any service execution is characterized as the following sequence of actions performing the corresponding OASIS concepts: object identification; introduction of event arguments; current state correctness; precondition satisfaction; valuation fulfillment; integrity constraint checking in the new state and trigger relationships test after a valid change of state. Various design decisions are described such as separating the interface of the generated prototype from its functionality. The possibility fo generating ActiveX components that can be executed and incorporated into defferent Windows transactional environments is mentioned. ActiveX is the newest version of Microsoft's OLE technology and enables applications to communicate with each other by means of messages passed with the aid of the OS. The previous version was called COM for Component Object Model. ActiveX adds features designed to enable the distribution of executable programs, called controls, via the internet. To use these controls, a computer must be running an OS that supports OLE, such as Window 3.1, 95, 98, NT or MacOS. Unlike Java applets, which run in a sandbox that protects the computer's file system, ActiveX controls can directly affect files. For this reason ActiveX controls are packaged with digitally signed certificates, which prove the program emanates from a respectable software publisher.

15) Gomez, J., et al., "*From Object-Oriented Conceptual Modeling to Component-Based Development*" Database and Expert Systems Applications. Lecture Notes in Computer Science (1677) pgs. 332-341 Springer-Verlag, 1999, ISBN: 3-540-66448-3; ISSN: 0302-9743, Florencia (Italia) August 30, 1999 [gomez99]:

16) Torres, I., "*Disseny i Implementació d'un Diccionari de Dades per a un Model Conceptual*" (In Valenciano), Master Thesis, June 2000.
Reference:
[Torres00MT]
Original title:
"Disseny i implementació d'un Diccionari de Dades per a un Model Conceptual Objectual"
Title translation:
"Design and implementation of a Data Dictionary for an Object-Oriented Conceptual Model"
Summary of contents:
This master thesis describes how to map the elements in the metamodel of a conceptual model to elements of a relational model. That is, how to persist in a relational database the information captured by a conceptual model.

Section One
It describes the state of the art for CASE Tools in general and for some of them in particular (System Architect, Rational Rose and Paradigm Plus, as well as the CASE OO-Method tool, which is the ancestor of the tool taught as one of the preferred embodiments of the patented invention).

Section Two
Enumerates the requirements for the work.

Section Three
Describes the information required for every conceptual primitive in a Conceptual Model (classes, attributes, relationships, preconditions, integrity constraints, triggers, ...etc). This is done by enumerating the properties of each metaclass (e.g: properties of the metaclass "Attribute" are "name", "attribute type", "attribute data type", "length", ...etc.)

Section Four
Teaches how metaclasses, metaattributes and metarelationships are mapped to tables, columns, primary keys and foreign keys.

Section Five
Describes the implementation of a C++ application that serves as client of the relational database repository so as to create, edit, query and delete the information it stores.

17) Pelechano, V., et al., "*An Automatic Code Generation Process for Dynamic Specialization Based on Design Patterns and Formal Techniques*", Actas de la IFIP International Conference on Software: Theory and Practice (ICS-2000), 16th IFIP World Computer Congress, pgs. 526-539, Pekin (China), August 2000; ISBN-7-5053-6100-4, Publishing House of Electronics Industry:
Teaches automatic code generation from conceptual models which have been reduced to a formal language specification. A conceptual model is first built that collects the desired properties of the program to be automatically written. That conceptual model is translated into a

formal language specification. Then an execution model is applies to the conceptual model. The execution model accurately states the implementation dependent features in order to represent the Conceptual Model in ad determined development environment. In other words, the execution model proposes a code generation strategy which obtains the representation of the modelling elements in a selected programming language according to a set of specific patterns. The execution model provides an architure for the solution by means of architectural patterns and a code generation strategy to obtain the software components of the architecture. A multitier architecture of interface tier, business tier and persistence tier is taught. A code generation strategy that allows obtaining the software components of the business tier in a systematic way is taught. The code generation strategy defines precise mappings between conceptual patterns and its represenation in code. The input to the process is the conceptual model made up of conceptual patterns based in OASIS cncepts.

18) Pastor, O., "*The OO Method Approach for Information Systems Modeling: From Object Oriented Conceptual Modeling to Automatic Programming*", Information Systems Journal, Elsevier Science, November 2001, Vol. 26/7, pgs. 507-534 [Pastor010]:
Teaches the OO Method of using OASIS concepts to develop a conceptual model containing an object view, a dynamic view and a functional model view, and as these models are constructed, a formal language specification is captured on the fly from the primitives defined in the models. Conventional UML graphical diagrams are used. The object model defines classes, attributes, services and class relationships (aggregation and inheritance) and agent relationships. Constraints which are formulas which define conditions objects in the class must satisfy are also specified. The dynamic model is used to specify object lives and interactions using a state transition diagram and an interaction diagram. Transitions and preconditions are defined, and it is taught that there is a state transition diagram for every class, but only one interaction diagram. The functional model is taught as capturing the semantics of any state change of an object as a result of execution of a service. Teaches use of a graphic modelling environment as a graphical editor for creation of the OASIS formal language specification. Valuations are defined as are derivations which alter the value of an attribute in terms of a derivation expression. Integrity constraints are defined as formulas which must be satisfied in every world (state). Preconditions are defined as formulas which must be satisfied before a state transition is allowed. Triggers are defined as services which are activated when the condition stated in the trigger formula holds. A template process for the execution model is introduced as identifying the user, obtaining the object system view, and service activation comprising: identifying the object server; introducing the service arguments; sending the message to the object server; checking the state transitions; checking preconditions; valuation fulfillment; integrity constraint checking in the new state and trigger relationship testing.

19) Molina, P., "*Especificación, de Interfaz de Usuario en OO-Method*" (In Spanish) Master Thesis, September 1998, DSIC/UPV, Valencia, Spain.
Reference:
[Molina98MT]
Original title:
"Especificación de interfaz de usuario en OO-Method"
Title translation:
"User interface specification in OO-Method"
Summary of contents:
This work is Pedro's master thesis and teaches a very primitive Presentation Model. In fact, the set of patterns described in this work are not regarded as part of a "Presentation Model" but

rather as an extension of the three original models (Objects, Dynamic and Functional)

Section One
Introduction to the work and its goals: augmenting OO-Method with some patterns that allow to capture user interface requirements so as to produce better code.

Section Two
Description of the state of the art at the time the master thesis was produced. Four works are described here:

- APT
    - Focused on the presentation of results of an interaction to the user of a system
- UIDE
    - Object-oriented environment to specify user interface requirements of a system from an abstract perspective
- FOCUS
    - Allows to use existing libraries that provide the functionality of a certain system (but have no user interface) and combine them with a layer that presents a user interface at runtime.
- TRIDENT
    - Allows to produce user interfaces from the description of tasks.

Section Three

Describes the OO-Method

First it introduces the formal specification language OASIS
Then it introduces the methodology as distinguishing between the Conceptual Model (the specification of "what" a system must do, comprising an Objects Model, a Dynamic Model and a Functional Model) and the Execution Model (the specification of "how" the elements of the Conceptual Model will behave at runtime).
An introduction to each of the three models (Objects, Dynamic and Functional) follows.
Then this section teaches that the elements of the Conceptual Model have a correspondence to elements in the OASIS formal specification language.

The section closes with a mention to an implementation of the methodology in the form of a CASE (modelling) tool.

Section Four
Describes the most commonly found user interaction techniques (as a result of observing which are the most frequent tasks a user carries out when interacting with a system):

- Introduction
- Selection
- Dependency
- Action Selection
- Information Display

Then discusses the importance of feedback to users when interacting with a system.

Section Five
Proposes how to specify in OO-Method each of the interaction techniques described in the previous section. So, for each interaction technique a pattern is proposed:
- Introduction Pattern
- Selection Pattern
- Dependency Pattern
- Action Selection Pattern
- Information Display Pattern

The description of each pattern comprises:
- Relevant information
- What element of OO-Method is augmented
- Representation of the relevant information in a metamodel
- Proposed user interface to input the relevant information in a tool

Section Six
Outlines how every abstract pattern could be implemented in a prototype (which could eventually be automatically generated). This is done in quite an informal way, without really providing details about concrete implementations nor defining a translation process or method.

Section Seven

Proposes a style guide for user interfaces (either manually or automatically obtained from a conceptual model).

Section Eight Implementation Cases

Describes a basic interface generator that produces code in Visual Basic for a prototype of a graphical user interface comprising:
- A menu grouping and organizing the services of the conceptual model
- A form for each service of the conceptual model

This basic generator is acknowledged to still be lacking the capability of fully supporting the patterns described in the work.

A case study closes this section proposing possible implementations in Visual Basic for each of the patterns described in the work.

Section Nine

Presents the conclusions and future works to be developed. Main conclusions argue the need for patterns to specify user interface requirements in an abstract way, while future works include the implementation of a translator that supports all the patterns.

20) Insfrán, E., et al., "*Ingeniería de Requisitos aplicada al modelado conceptual de interfax de usuario*" (In Spanish), In Procs. Of IDEAS'2001, Santo Domingo, Heredia, Costa Rica, CIT, pgs. 181-192, April, 2001.

21) Molina, P., et al., "*Specifying Conceptual Interface Patterns in an Object-Oriented Method with Code Generation*", In Proceedings of User Interfaces for Data Intensive Systems, UIDIS'2001, Zurich, Switzerland, IEEE Computer Society, pgs. 72-79, May 31, 2001 [Molina01] (this reference is not prior art to the parent patent **6,681,383** and is not prior publication prior art to any patent application filed before the publication date hereof):

Describes the OO Method of creating a conceptual model including a business model with an object model, dynamic model and function model and introduces the concept of a presentation model. Then an execution model fixes the characteristics of the final software product in terms of user interface, access control, service activation and all the other implementation dependent properties. The formal language specification is obtained in an automated way from the conceptual model, and the execution model automatically builds a prototype that is the functional equivalent of the conceptual model. Teaches that code generators for C++, Visual Basis combined with Microsoft Transactional Server and Java have already been implemented. Introduces the concept of conceptual interface patterns. Introduces the following patterns for interfaces found in the literature and industrial projects: 1) pattern of introduction which appears whenever the system needs data to be introduced into it by the user; 2) patterns of selection which appears when the user is to be given the opportunity to enter data by selecting from a list; 3) patterns of dependency which arise when the determination of a value of one or more arguments leads to the variation of state (value or restrictions) in other arguments or items of data requested by the system; 4) patterns of action selection which are presented when a system requires user intervention to perform a task and needs to present actions a user can choose to invoke (used to supplant the command line interface); and 5) information presentation patterns where the results of processing are presented to the user. Notes that conventional UML based software production methods rarely take into account the particular features of the interface specifications when facing the conceptual modelling phase. Mentions three prior art methods of deriving user interface from specifications of the application domain: Genius; TRIDENT and Janus.

22) Molina, P., et al., "*Prototipado rápido de interfaces de usuario*", (In Spanish), In Procs. Of IDEAS'2002, La Habana, Cuba, pgs. 78-90, April 23, 2002 [Molina02a]:

Reference:

[Molina02a] This paper is not prior art to the parent patent 6,681,383 or any of the CIP applications having serial numbers: 09/872,413 filed 6/1/01; 09/872,087 filed 6/1/01 or 09/872,333 filed 6/1/01 nor to any other case filed after the CIPs and assigned to the assignee thereof. It is cited here for information purposes only and for the art is cites in its footnotes.

Original title:

"Prototipado rápido the interfaces de usuario"

Title translation:

"Rapid prototyping of user interfaces"

Summary of contents:

This paper describes the use of inference techniques so as to create default patterns in the Presentation Model which allow rapid prototyping of user interfaces.

Section One

Describes OO-Method as comprising four models: Objects Model, Dynamic Model, Functional Model and Presentation Model.

Section Two

Provides more detail about the Presentation Model and categorizes patterns in three levels:
- Hierarchical Actions Tree
- Interaction Units
    - o Service Interaction Unit
    - o Instance Interaction Unit
    - o Population Interaction Unit
    - o Master/Detail Interaction Unit
- Auxiliary Patterns
    - o Introduction
    - o Defined Selection
    - o Supplementary Information
    - o Dependency
    - o Arguments Grouping
    - o Filter
    - o Display Set
    - o Navigation
    - o Actions
    - o Order Criteria

Section Three

Describes techniques to infer:
- Views
- Hierarchical Actions Tree
- Interaction Units
    - o Service Interaction Units
        - ß Inference of: Alias
        - ß Inference (for each service argument) of:
            - • Alias
            - • Introduction Pattern applied

• Defined Selection Pattern applied
• Supplementary Information Pattern applied
• Population Interaction Unit applied
o Instance Interaction Units
ß Inference of: Alias
o Population Interaction Units
ß Inference of :
• Alias
• Filter Alias
• Inference (for each filter variable) of:
o Supplementary Information Pattern applied
o Population Interaction Unit applied
o Master/Detail Interaction Units
ß Inference of: Alias
- Auxiliary Patterns
o Display Set
ß Inference (for each element of it) of:
• Alias
• Introduction Pattern applied
• Defined Selection Pattern applied
o Navigations
ß Inference (for each element of it) of:
• Alias
o Actions
ß Inference (for each element of it) of:
• Alias

Section Four
Describes a case study where these techniques were applied

Section Five
States that inference techniques have been implemented in the modelling tool and in the transformation engines (translators).

Section Six
Relates this paper to previous works

Section Seven
Concludes that the use of these inference techniques aids in rapidly prototyping user interfaces.

23) Molina, P., et al., "*JUST-UI: A User Interface Specification Model*" In Computer-Aided Design of User Interfaces III, Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002, Kluwer Academics Publisher, Dordrecht, pgs. 63-74, Valenciennes, France, May 15, 2002 [Molina02c]:

This paper is not prior art to the parent patent 6,681,383 or any of the CIP applications having serial numbers: 09/872,413 filed 6/1/01; 09/872,087 filed 6/1/01 or 09/872,333 filed 6/1/01 nor to any other case filed after the CIPs and assigned to the assignee thereof. It is cited here for information purposes only and for the art is cites in its footnotes.

This paper teaches elemental user interface requirements such as guide the underlying concept of most user interfaces: how to search? how to order? what to see? and what to do?

The paper recognizes that the prior art tools such as Rational Rose, Together and Argo do not explicitly consider the specification of user interfaces. Thus, the specification of the software to be built with these tools is both very complex and incomplete.

The approach taught in this paper like the inutitive approach taught by Constantine et al. the UI is to divine a desired navigation between interaction units such as forms or web pages using arrows between the interaction units. The approach to specify a UI taught in this paper extends the OO Method of generating a conceptual model using the models taught elsewhere herein. The final goal is to gather user interface requirements in an abstract way and to automatically generate the final user interface to different target environments such as Web, Windows, X11, UMTS or PDAs. These user interface requirements are gathered in the Presentation Model part of the Conceptual Model. The user interface requirements are specified in terms of patterns as follows:

1) Filter: a pattern useful for searching for objects in a class that satisfy a given condition.
2) Order Criterion: after gathering search requirements, the analyst inquires how must object be ordered: alphabetically, numerically etc.
3) Display set: after the objects are selected using the filter and ordered, what properties does the user need to see from a given object.
4) Navigation: information about whether the user needs to view any additional information related to the object selected using the filter, order and display set patterns, e.g. if the object found is a rental car, other related information might include the contract terms and prices.
5) Actions: actions are User Interface possibilities which allow a user to invoke actions to change the state of the current object.

Complex patterns can be built using Presentation Units. A Service Presentation Pattern is used to model a dialogue to deal with a service. The user must provide the arguments and launch the service. An Instance Presentation Pattern is used to model the data presenation of an instance and support interaction with it. It is oriented to object manipulation and arises out of the need to view single objects. The analyst must gather information about which attributes of an object does the user wish to view and what actions can be done on the object and which additional information may be needed about the object needs to be reached for the interface via navigation. A Population Presentation Pattern models units of interaction focused on showing sets of instances of a class. This pattern deals with the necessity of working with object collections. A Master-Detail Presentation Pattern deals with master-slave presentations where an aggregation relationship between classes is crossed from the master to slave direction. When the master component changes, the slave components do also. For example, and invoice as the master object and the detailed line items as the slave objects yielding details about the invoice represent a master-slave presentation pattern.

24) Molina, P., et al., "*User Interface Conceptual Patterns*", In Proceedings of the 4th International Workshop on Design Specification & Verification of Information Systems DSV-IS'2002, Rostock, Germany, pgs. 201-214, June 4, 2002 [Molina 02d]:

This paper is not prior art to the parent patent 6,681,383 or any of the CIP applications having serial numbers: 09/872,413 filed 6/1/01; 09/872,087 filed 6/1/01 or 09/872,333 filed 6/1/01 nor to any other case filed after the CIPs and assigned to the assignee thereof. It is cited here for information purposes only and for the art is cites in its footnotes.

See Figure 1 for a diagram of user interface patterns and their subcomponents. Introduces the notion of a heirarchical action tree having an IU Service component, a UI population component, an UI instance component and a UI master-detail component. All components have subcomponents. The capture of each UI pattern in the modeler tool is accomplished using a form template which is filled in by the analyst to instantiate the pattern.

25) Ingenieria de requisitos aplicada al modelado conceptual de interfacz de usuario, April 3, 2001 [insfran01mp]:

Reference:
[Insfran01mp]
Original title:
"Ingeniería de requisitos aplicada al modelado conceptual de interfaz de usuario"
Title translation:
"Requirements engineering applied to conceptual modelling of user interfaces"
Summary of contents:
This paper introduces a method to obtain the features of a user interface from a Requirements Model. Said Requirements Model is, in terms of abstraction levels, above Conceptual Models, and it captures properties from three complementary points of view: functions (which correspond to external interactions), communication (between functions and actors) and behaviour (description of atomic actions).

Section Two
This section describes how the OO-Method is augmented with a Presentation Model and a Requirements Model.

The OO-Method is described as comprising three models: Objects Model, Dynamic Model and Functional Model, but this paper proposes that it be extended with two more models: the Presentation Model and the Requirements Model. Therefore a Conceptual Model in OO-Method encompasses the three original models (Objects, Dynamic and Functional) plus the Presentation Model, while the Requirements Model stands alone and is one level of abstraction above Conceptual Model

The Requirements Model comprises three parts:
- Mission Statement
    - Describes the general goal of the system
- Functions Refinement Tree
    - Decomposes the functionality of the system in external interactions which are organized in a hierarchical fashion as a result of refining the Mission Statement.
- Use Case Model
    - Includes the Use Cases Specification to describe the decomposition of external interactions and a Use Case Diagram to show the communication between actors and the system.

The paper points out that the Conceptual Model can be obtained from the Requirements model, specifically from the detailed specification of external interactions by performing some Requirements Analysis Process (which is based on Use Case Specifications and Sequence Diagrams).
The Presentation Model comprises the following patterns:
- Patterns for Action Selection
    - Actions Hierarchy
    - Navigations
    - Actions

- Patterns for Presentation
    - Service Presentation
    - Instance Presentation
    - Class Population Presentation
    - Master/Detail Presentation
- Auxiliary Patterns
    - Introduction
    - Defined Selection
    - Population Selection
    - Supplementary Information
    - Dependency

Section Three

Describes a method to obtain the Actions Hierarchy (in the Presentation Model) from the Functions Refinement Tree (in the Requirements Model)

Section Four
Presents the conclusions. This paper only illustrates how to obtain part of the Presentation Model (the Actions Hierarchy, currently referred to as "Hierarchical Actions Tree") from a Requirements Model. It does not teach how to obtain the rest of patterns that make up the Presentation Model from a Requirements Model, it only states that this is part of future research.

26) Eager, et al., U.S. Patent 5,960,200, Filed Sep. 16, 1996, Date of Patent Sep. 28, 1999.

27) Scandura, U.S. Patent 6,275,976 B1, Filed Feb. 25, 1997, Date of Patent Aug. 14, 2001.

THE OFFICE ACTIONS INCLUDED HEREWITH

CHG-001 (now US patent 6,681,383): generall claims an entire system including a graphical front end to create a formal languge specification, a validator to validate the specification, and a translator to translate the validated specification into working code. Three office actions are included herewith dated: May 25, 2001; April 4, 2003; and October 23, 2002.

The PCT and foreign counterparts of CHG-001 have generated one IPER dated 7/3/03, an International Search Report dated 6/13/02 and one and only one foreign office action from Australia dated 6/1/2004.

CHG-001.1P (CIP #1) up to this point generally claims a user interface to present tools by which primitives can be defined, at least some of which are displayed graphically, to create a conceptual model of software to be written which is then converted to a formal language specification. The conceptual model is comprised of an object model, a functional model and a dynamic model. This formal language specification is then validated to ensure it it is complete and correct. This case has generated four office actions included herewith dated: October 7, 2004; June 7, 2005; February 26, 2004; and February 2, 2005.

CHG-001.2P (CIP #2) up to this point generally claims a graphical user interface to present tools by which primitives can be defined to create a conceptual model of software to be written which is then converted to a formal language specification. The conceptual model is comprised of an object model, a functional model and a dynamic model. Some of the dependent claims include validation. This case has generated three office actions included herewith dated: April 25, 2005; August 16, 2004; April 22, 2004.

CHG-001.3P (CIP #3) up to this point generally claims a method and apparatus for validating a formal language specification and then automatically translating it into working code using an execution model. This case has generated three office actions included herewith dated: July 9, 2004; January 26, 2005 and July 14, 2005.

Respectfully submitted,

[illegible]

Ronald Craig Fish
Reg. No. 28,843
Attorney for Applicant(s)

Dated: 12/15/05

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Commissioner of Patents and Trademarks, Washington D.C. 20231 on _____
(Date Of Deposit)

[illegible]

Ronald Craig Fish, President
Ronald Craig Fish a Law Corporation
Reg. No. 28,843

OIPE IAP15
DEC 2 7 2005
PATENT & TRADEMARK OFFICE

SHEET 1 OF 3

FORM PTO-1449 (Rev. 2-32)	U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE	ATTY. DOCKET NO. CHG-001.3P	SERIAL NO. 09/872,333
INFORMATION DISCLOSURE STATEMENT BY APPLICANT		APPLICANT PASTOR, et al.	
(USE SEVERAL SHEETS IF NECESSARY)		FILING DATE 6/1/01	GROUP 2124

U.S. PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB CLASS	FILING DATE IF APPROP.
	A	5,960,200	Sep. 28, 1999	Eager, et al.	395	705	Sep. 16, 1996
	B	6,275,976 B1	Aug. 14, 2001	Scandura	717	1	Feb. 25, 1997
	C						
	D						
	E						
	F						
	G						
	H						
	I						
	J						

FOREIGN PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	PUB. DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
							YES	NO
	K							
	L							
	M							
	N							
	O							
	P							
	Q							
	R							

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

EXAMINER INITIAL		
	S	Letelier, P., et al., "*OASIS Version 3.0: Un Enfoque Formal Para el Modelado Conceptual Orientado a Objectos*" (In Spanish), ISBN: 84-7721-663-0, Legal Diposit: V-3484-1998, Servicio de Publicaciones de la UPV, SP-UPV 98-4011, Valencia, Spain, 1998.
	T	Pelechano, V., "*OO-Method: Implementación de un Entorno Gráfico para el Análisis y Diseño de Sistemas De Información OO*" (In Spanish), Master Thesis, 1994.
	U	Pelechano, V., et al., "*CASE OO-Method: Un Entorno de Producción Automática de Software*" (In Spanish) Actas de la Convenció Informática Latina CIL-95, Barcelona, June, 1995.

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if citation considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

OIPE IAP15
DEC 2 7 2005
PATENT & TRADEMARK OFFICE

SHEET 2 OF 3

FORM PTO-1449 (Rev. 2-32)	U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE	ATTY. DOCKET NO. CHG-001.3P	SERIAL NO. 09/872,333
INFORMATION DISCLOSURE STATEMENT BY APPLICANT		APPLICANT PASTOR, et al.	
(USE SEVERAL SHEETS IF NECESSARY)		FILING DATE 06/01/2001	GROUP 2124

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

EXAMINER INITIAL		
	A	Romero, J., "*Diseño de un Entorno de Producción de Software basado en el Lenguaje de Especificación OASIS y en la utilización de PowerBuilder como Herramienta de Desarrollo Gráfica y C/S*" (In Spanish), Master Thesis, Valencia, March, 1996.
	B	Pastor, O., et al., "*An Object Oriented Methodological Approach for Making Automated Prototyping Feasible*", Database and Expert Systems Applications. Lecture Notes in Computer Science (1134) pgs. 29-39 Springer-Verlag. 1996, ISBN: 3-540-61656-x, ISSN: 0302 9743, Zurich (Suisse).
	C	Pelechano, V., et al., "*Implementación y comprobación de restricciones de integridad dinámicas en entornos de programación orientados a objectos*" (In Spanish), II Jornadas Nacionales de Ingeniería de Software, Universidad el País Vasco, San Sebastián, 3-5 Septiembre 1997, pgs. 101-117.
	D	Pastor, O., et al., "*Linking Object-Oriented Conceptual Modeling with Object-Oriented Implementation in Java*", VIII Conference on Database and Expert Systems Applications, (DEXA'1997), ISGN: 3-540-63478-9, LNCS (1308), Toulouse, France, 1997.
	E	Pastor, O., et al., "*OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods*", 9th International Conference on Advanced Information Systems Engineering, (CaiSE'1997) ISGN: 3-540-63107-0, LNCS (1250), Barcelona, Spain, 1997.
	F	Pastor, O., et al., "*Object Oriented Conceptual Modeling Techniques to Design and Implement a Sound and Robust Oracle Environment*" Actas de Oracle OpenWorld 97, Viena (Austria) 7-11 Abstract publicado en Oracle OpenWorld Review pg. 36, April, 1997.
	G	Romero, J., et al., "*Una Herramienta de Generación Automática de Software*" (In Spanish) In Procs of IDEAS-98 - I Workshop Iberoamericano en Ingeniería de Requisitos y Ambientes Software, Porto Alegre, Brasil, April, 1998.
	H	Gomez, J., et al., "*The Execution Model: A Component-Based Arquitecture to Generate Software Components from Conceptual Models*" In Procs of International Workshop on Component-based Information Systems Engineering, 10th International Conference on Advanced Information Systems Engineering, CAiSE-98 Pisa (Italia), pgs. 87-94, ISSN 1170-487X.
	I	Pastor, O., et al., "*From Object Oriented Conceptual Modeling to Automated Programming in Java*", Conceptual Modeling — ER'98, Lecture Notes in Comptuer Science (1507), pgs. 183 197, Springer-Verlag, 1998, ISBN: 3-540-65189-6, ISSN: 0302-9743, Singapur.
	J	Pastor, O., et al., "*Mapping Aggregation from Object-Oriented Conceptual Modeling to Object Oriented Programming*", In Procs of Third International Conference on Object-Oriented Technology, WOON-98, pgs. 59-70, San Petersburgo, Russia, July, 1998.
	K	Romero, J., et al., "*Automatic Object-Oriented Visual Programming with OO-METHOD*", Software and Hardware Engineering for the 21th Century, pgs. 345-354, World Scientific and Engineering Society Press, ISBN: 960-8052-06-8.

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if citation considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

OIPE IAP15
DEC 2 7 2005
PATENT & TRADEMARK OFFICE

SHEET 3 OF 3

FORM PTO-1449 (Rev. 2-32)	U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE	ATTY. DOCKET NO. CHG-001.3P	SERIAL NO. 09/872,333
	INFORMATION DISCLOSURE STATEMENT BY APPLICANT	APPLICANT PASTOR, et al.	
	(USE SEVERAL SHEETS IF NECESSARY)	FILING DATE 06/01/2001	GROUP 2124

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

EXAMINER INITIAL		
	A	Gomez, J., et al., "*From Object-Oriented Conceptual Modeling to Component-Based Development*" Database and Expert Systems Applications. Lecture Notes in Computer Science (1677) pgs. 332-341 Springer-Verlag, 1999, ISBN: 3-540-66448-3; ISSN: 0302-9743, Florencia (Italia).
	B	Torres, I., "*Disseny i Implementació d'un Diccionari de Dades per a un Model Conceptual*" (In Valenciano), Master Thesis, 2000.
	C	Pelechano, V., et al., "*An Automatic Code Generation Process for Dynamic Specialization Based on Design Patterns and Formal Techniques*", Actas de la IFIP International Conference on Software: Theory and Practice (ICS-2000), 16th IFIP World Computer Congress, pgs. 526 539, Pekin (China), Agosto 2000; ISBN-7-5053-6100-4, Publishing House of Electronics Industry.
	D	Pastor, O., "*The OO Method Approach for Information Systems Modeling: From Object Oriented Conceptual Modeling to Automatic Programming*", Information Systems Journal, Elsevier Science, October 2001, Vol. 26/7, pgs. 507-534.
	E	Molina, P., "*Especificación, de Interfaz de Usuario en OO-Method*" (In Spanish) Master Thesis, September 1998, DSIC/UPV, Valencia, Spain.
	F	Insfrán, E., et al., "*Ingeniería de Requisitos aplicada al modelado conceptual de interfax de usuario*" (In Spanish), In Procs. Of IDEAS'2001, Santo Domingo, Heredia, Costa Rica, CIT, pgs. 181-192, April, 2001.
	G	Molina, P., et al., "*Specifying Conceptual Interface Patterns in an Object-Oriented Method with Code Generation*", In Proceedings of User Interfaces for Data Intensive Systems, UIDIS'2001, Zurich, Switzerland, IEEE Computer Society, pgs. 72-79, Mary, 2001.
	H	Molina, P., et al., "*Prototipado rápido de interfaces de usuario*", (In Spanish), In Procs. Of IDEAS'2002, La Habana, Cuba, pgs. 78-90, April, 2002.
	I	Molina, P., et al., "*JUST-UI: A User Interface Specification Model*" In Computer-Aided Design of User Interfaces III, Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002, Kluwer Academics Publisher, Dordrecht, pgs. 63-74, Valenciennes, France, May, 2002.
	J	Molina, P., et al., "*User Interface Conceptual Patterns*", In Proceedings of the 4th International Workshop on Design Specification & Verification of Information Systems DSV-IS'2002, Rostock, Germany, pgs. 201-214, June, 2002.
	K	

EXAMINER	DATE CONSIDERED

EXAMINER: Initial if citation considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.